VERKKOSOVELLUKSEN TOTEUTUS FIREBASEN JA ANGULARJS:N AVULLA

Reini Valtanen

Opinnäytetyö
Joulukuu 2015
Tietojenkäsittelyn koulutusohjelma

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

VALTANEN, REINI
Verkkosovelluksen toteutus Firebasen ja AngularJS:n avulla

Opinnäytetyö 30 sivua, joista liitteitä 2 sivua
Joulukuu 2015

Opinnäytetyön tavoitteena oli tutkia Firebasen käyttöä sekä sen soveltuvuutta nykyaikaiseen verkkosovellukseen. Tarkoituksena oli toteuttaa sähköisen kanban-taulun kaltainen työkalu toimeksiantaja Mainostoimisto Värikäs Oy:n työntekijöiden henkilökohtaiseen projektienhallintaan. Sovelluksen tuli olla responsiivinen, jotta sitä voisi käyttää eri laitteilla näyttökoosta riippumatta.

Sovelluksen frontend toteutettiin AngularJS-frameworkilla ja backend Firebasella. Toteutuksessa käytettiin iteratiivista kehittämismallia.

Sovelluksen toteutus oli onnistunut ja vastasi toimeksiantajan toiveita. Firebasen todettiin pienentävän kehittäjän työmäärää huomattavasti tarjoamalla monia verkkosovelluksen toiminnallisuuksia valmiina.

Sovellus otettiin käyttöön toimeksiantajalla keväällä 2015.

Asiasanat: firebase, angularjs

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems

VALTANEN, REINI
Building a Web Application with Firebase and AngularJS

Bachelor's thesis 30 pages, appendices 2 pages
December 2015

The object of this thesis was to see how to use Firebase and how it suits modern web application development. The purpose was to develop an electric kanban board for the client, Mainostoimisto Värikäs. The application would be used as employees' personal project management tool.

The front end of the application was built using AngularJS framework, and Firebase was used to build the back end.

The development process was successful. Firebase was found to be very useful in terms of decreasing required development time. It provided many functionalities out of the box.

Key words: firebase, angularjs

SISÄLLYS

1 JOHDANTO 6
2 OPINNÄYTETYÖN TAUSTA 7
2.1 Tavoite ja tarkoitus 7
2.2 Toimeksiantaja 7
2.3 Firebase 8
2.3.1 Firebase Realtime Database 8
2.3.2 Firebase Dashboard 9
2.3.3 Palvelimet ja tiedonsiirto 11
2.3.4 Alustat ja kirjastot 11
3 SOVELLUKSEN SUUNNITTELU 13
3.1 Käyttöliittymä 13
3.2 Rakenne 14
3.3 Tekniikat 15
4 SOVELLUKSEN TOTEUTUS 17
4.1 Sovelluksen runko 17
4.2 Käyttöliittymä 17
4.2.1 Päänäkymä 18
4.2.2 Tehtävien muokkaus 19
4.2.3 Käyttäjän tietojen muokkaus 20
4.3 Firebasen toiminnallisuus 20
4.3.1 Tietojen haku ja näyttö 20
4.3.2 Tietojen tallennus 21
4.4 Kirjautuminen 21
4.5 Palvelintila 23
4.6 Osioiden yhdistäminen 24
4.7 Testaus 25
5 POHDINTA 27
LÄHTEET 28
LIITTEET 29
Liite 1. Ohjeet Firebasen ja AngularFiren käyttöönottoon ja peruskäyttöön 29

LYHENTEET JA TERMIT

AngularFire	Firebasen AngularJS kirjasto.
AngularJS	Googlen ylläpitämä avoimen lähdekoodin JavaScript-framework.
Bootstrap	Kokoelma avoimen lähdekoodin työkaluja frontend-kehitykseen.
Bower	Pääasiassa frontend-kehitykseen tarkoitettu pakettienhallintatyökalu.
Framework	Suom. ohjelmistokehys. Muodostaa käytettävän sovelluskielen päälle valmiiksi rakennettuja ohjelman osia helpottamaan kehittäjän työtä.
Git	Hajautettu versionhallintaohjelmisto.
JSON	Avoimen standardin tiedostomuoto tiedonvälitykseen.
LAMP	Kokoelma avoimen lähdekoodin ohjelmia, joista muodostetaan palvelin, jolla voidaan suorittaa dynaamisia www-sivuja. LAMP muodostuu sanoista Linux, Apache, MySQL ja PHP/Perl/Python.
MEAN	Kokoelma JavaScriptilla toteutettuja avoimen lähdekoodin ohjelmia, joista muodostetaan palvelin verkkosovelluksille. MEAN muodostuu sanoista MongoDB, ExpressJS, AngularJS ja Node.js.
Node.js	Avoimen lähdekoodin JavaScript-ajoympäristö palvelinpuolelle.
NoSQL	Normaalista relaatiomallista poikkeava tietokanta, joka ei seuraa ennalta määritettyä skeemaa, vaan sallii skaalautuvuuden tarpeen mukaan. NoSQL on myös erittäin nopea suoritettaessa suurta määrää samanaikaisia luku- ja kirjoitusoperaatioita.
Npm	Node.js:n pakettienhallintatyökalu.
Sass	Sovelluksen tyylittelyyn käytettävä kieli.

1 JOHDANTO

Verkkosovelluskehityksen nousevana trendinä viime vuosina ovat olleet sovellukset, jotka ominaisuuksiltaan ja käytettävyydeltään vastaavat natiiveina ohjelmina tietokoneelle asennettuja ohjelmia. Perinteinen LAMP-mallin (Linux, Apache, MySQL, PHP) palvelinympäristö ei enää ole itsestäänselvyys valittaessa verkkosovelluksen tekniikoita. MEAN-mallin ympäristöllä toteutettuja sovelluksia on aina vain enemmän. MEAN-mallissa (MongoDB, ExpressJS, AngularJS ja Node.js) jokainen sovelluksen osa on toteutettu JavaScriptilla. Näin LAMP-mallin kieliyhdistelmän sijasta saadaan sovellus toteutettua yhtä kieltä käyttämällä.

Firebase vie sovelluskehityksen vielä yhden askeleen eteenpäin tarjoamalla palvelua, joka korvaa mahdollisesti koko verkkosovelluksen taustalogiikan. Tässä opinnäytetyössä selvitän, miten Firebase soveltuu verkkosovelluskehitykseen. Toteutan Firebasella sovelluksen, joka vielä jokin aika sitten olisi suurella todennäköisyydellä toteutettu LAMP-mallin ympäristön päälle.

Opinnäytetyöni rakentuu tavoitteen, tarkoituksen ja toimeksiantajan esittelystä, seuraavassa luvussa keskitytään Firebaseen ja sen ominaisuuksiin. Tätä seuraa luvut sovelluksen suunnittelusta ja toteutuksesta. Lopussa oleviin liitteisiin olen kerännyt ohjeet Firebasen käyttöönottoon.

2 OPINNÄYTETYÖN TAUSTA

2.1 Tavoite ja tarkoitus

Tavoitteenani on selvittää, miten Firebase sopii sovelluskehitykseen ja miten sitä käytetään sellaisessa sovelluksessa, joka vielä jokin aikaa sitten olisi lähes poikkeuksetta toteutettu käyttäen LAMP-ympäristöä. LAMP muodostuu sanoista Linux, Apache, MySQL ja PHP/Perl/Python ja tarkoittaa kokoelmaa avoimen lähdekoodin ohjelmia, joista muodostetaan palvelin, jolla voidaan suorittaa dynaamisia www-sivuja.

Tarkoituksena on toteuttaa sähköisen kanban-taulun kaltainen työkalu yksittäisen työntekijän omien työtehtävien hallintaan. Kanban-taulu on työväline töiden visualisointiin, jossa työn eri vaiheet määritellään omiksi sarakkeiksi (Ashmore 2014). Yksinkertaisimmillaan kanban–taulu on tussitaulu, johon on piirretty sarakkeet ja työt on kirjoitettu muistilapuille ja järjestetty sarakkeisin.

Nykyajan vaatimusten mukaisesti sovellus on responsiivinen, eli sama sovellus muokkaantuu sopimaan päätelaitteen näytölle sen koosta riippumatta. Sovellus toteutetaan käyttäen Firebasen ja sen AngularFire-kirjaston lisäksi AngularJS-frameworkia sekä sen liitännäisiä. Responsiivisuuden toteuttamiseen käytetään Bootstrapia.

2.2 Toimeksiantaja

Toimeksiantajana työlle toimii tamperelainen Mainostoimisto Värikäs Oy. Värikäs on sekä perinteisen printtimedian että digitaaliseen markkinointiviestinnän osaaja. Työntekijöitä Värikkäässä on 10.

Sovellukselle on toimeksiantajalla selvä tarve. Tällä hetkellä työntekijät kirjoittelevat erinäisille lapuille omia projektejaan sekä niiden tilanteita ja muistiinpanoja. Usein laput katoavat ja tilanne on välillä vähintään sekava.

Sovelluksella pystytään korvaamaan erinäiset laput täysin. Työpisteellä selaimella kirjattuja tietoja pystyy myös helposti lukemaan ja muokkaamaan palavereissa mukana

olevalla kännykällä tai tabletilla. Työntekijä merkitsee omat työnsä uusina tehtävinä, antaa niille otsikon, kuvauksen ja mahdollisti päivämäärän, jolloin tehtävän pitää olla valmis.

Toimeksiantajan edustajat olivat mukana suunnittelemassa sovelluksen toiminnallisuutta ja käyttöliittymää.

2.3 Firebase

Firebase Inc. on syyskuussa 2011 San Franciscossa perustettu pilvipalveluita tarjoava yritys, jonka Google osti lokakuussa 2014.

Firebase tarjoaa BaaS-palvelua, josta se käyttää nimeä *The Realtime App Platform*. BaaS tulee termistä *backend as a service* ja tarkoittaa verkko- ja mobiilisovelluksen taustalogiikan pilvipalvelua (Vinci 2014).

Firebasen palvelu sisältää kehittäjiä helpottavia alustariippumattomia työkaluja ja ominaisuuksia, kuten tietovaraston, palvelimen sovellukselle ja kirjautumislogiikan muun muassa sosiaalisen median tunnuksilla.

2.3.1 Firebase Realtime Database

Firebasen tietovarasto on NoSQL-tietokantamalliin perustuva tietovarasto, jossa käyttäjän tiedot tallennetaan standardin mukaisessa JSON-formaatissa. Itsessään tietokannan tekniikka tai tapa, jolla se tiedot varastoi ei ole mitään mullistavaa. NoSQL poikkeaa kuitenkin relaatiopohjaisesta MySQL-tietokantamallista siinä määrin, että kehittäjältä saattaa mennä hetki totutusta poikkeavan ajatusmallin ymmärtämiseen.

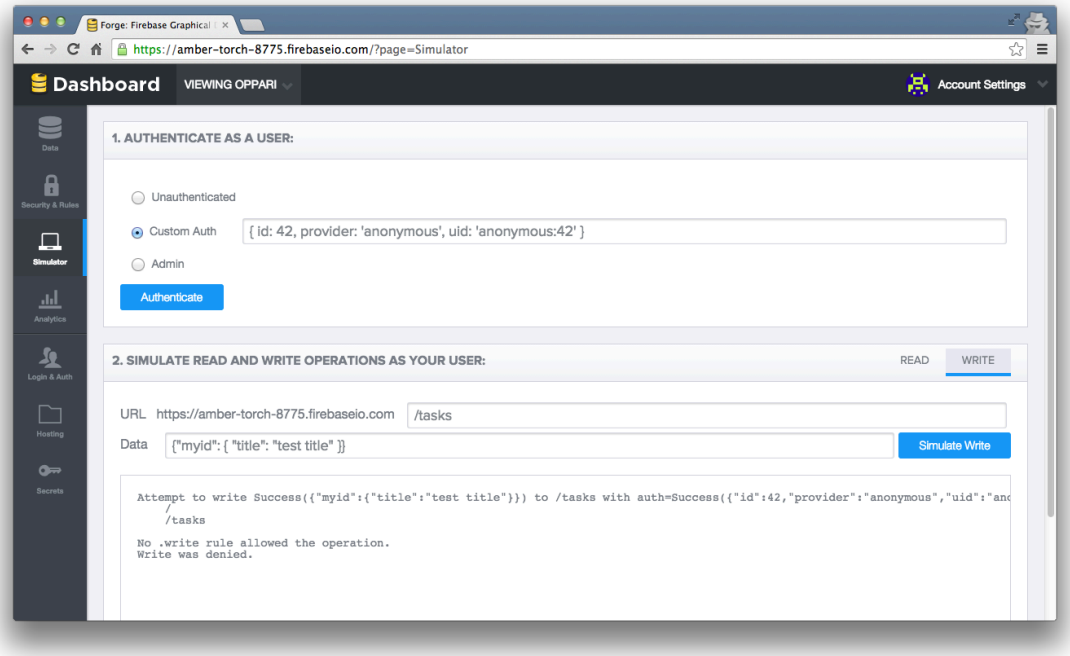
Varsinainen etu kehittäjälle tulee Firebasen tarjoamasta taustalogiikasta, jolla voidaan parhaimmillaan saadaa korvattua sovelluksen perinteinen taustalogiikka kokonaan. Tietenkään Firebasen käyttö ei millään tavalla estä perinteisempiäkään ratkaisuita, vaan Firebasea pystyy käyttämään osana mitä tahansa järjestelmää REST-rajapinnan kautta.

Joissain sovelluksissa käyttäjälle halutaan tuoda automaattisesti uutta sekä mahdollisesti muuttunutta aikaisempaa tietoa. Tähän on aikaisemmin ollut muutamia eri keinoja. Perinteinen ja kehittäjän kannalta yksinkertaisesti toteutettava tapa on määrittää sovellus lähettämään kyselyitä taustajärjestelmälle tietyin aikavälein. Tätä tapaa kutsutaan termillä *polling* (Mehta 2014). Toinen, hieman kehittyneempi tapa on rakentaa websocket, joka tarkoittaa kaksisuuntaista yhteyttä esimerkiksi selaimen ja palvelimen välillä (Harrop, Ho & Schaefer 2014).

Firebase tarjoaa myös tietojen synkronointiin valmiin ratkaisun. Käyttämällä sen kirjastoja tiedonhakuun sovelluksessa saadaan tiedon muuttuessa myös sovelluksessa näytettävä tieto muuttumaan automaattisesti. Tämä siis koskee niin lisättyä, muokattua kuin poistettuakin tietoa kaikilla Firebasen tarjoamilla kirjastoilla. Ainostaan REST-rajapinnan kautta haettavaa tietoa ei automaattisesti pystytä synkronoimaan, vaan silloin kehittäjän tulee käyttää vaihtoehtoista tapaa.

2.3.2 Firebase Dashboard

Firebase on tehnyt sovelluskohtaisesta tietovaraston hallinnasta ja ylläpidosta helppoa graafisessa ylläpitotyökalussaan, Firebase Dashboardissa (ks. kuva 1). Dashboardissa on mahdollista hallita tallennettua dataa sekä turvallisuus- ja kirjautumisasetuksia. Muita Dashboardin toiminnallisuuksia ovat oikeuksien simulointi, sovelluksen analytiikan tarkastelu ja Hosting-palvelun deploy-historia, joka on on lokitieto sovelluksen julkaistuista versioista.

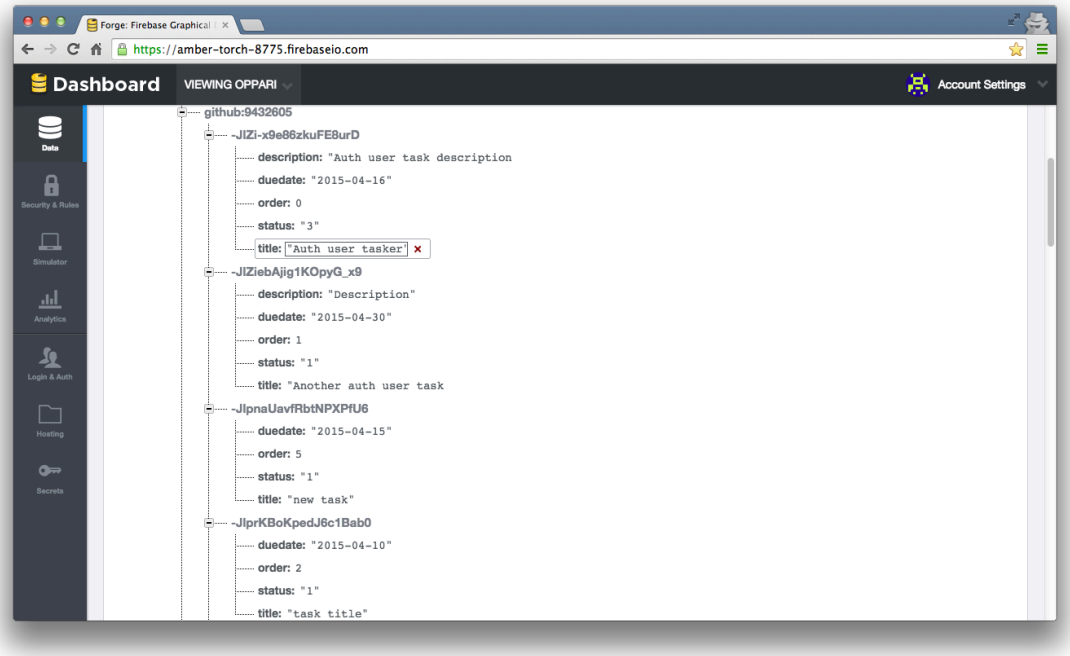


KUVA 1. Firebase Dashboard Simulator

Oikeuksien simuloinnilla voidaan testata turvallisuusasetuksia simuloimalla eri käyttäjätunnuksia tai esiintymällä sovelluksen adminina. Tällä työkalulla on hyvä testata turvallisuusasetusten oikeellisuutta ja toimivuutta.

Tiedonhallinnassa tallennettua dataa pystyy tarkastelemaan visuaalisessa muodossa (ks. kuva 2). Tallennettu JSON on hierarkkisessa järjestyksessä ja eri tasoja pystyy avaamaan ja piilottamaan. Tietoja pystyy myös lisäämään, muokkaamaan ja poistamaan. Muualla tapahtuva datan muutos näytetään tässä näkymässä korostetusti eri väreillä. Koko datan pystyy myös tallentamaan JSON-muodossa omalle koneelleen. Oman datan vienti Firebaseen on myös mahdollista Dashboardin kautta.

Analytiikka-osiossa näytetään graafisesti sovelluksen käyttötilastoja tallennetun ja siirretyn datan sekä yhtäaikaisten käyttäjien osalta. Tilastoja näytetään viimeisimpien 24 tunnin sekä 30 vuorokauden ajanjaksoilta. Tilastot päivittyvät 15-20 minuutin välein, joten aivan reaaliaikaista tietoa ei ole.



KUVA 2. Kuvankaappaus Firebasen tiedonhallinnan työkalusta.

2.3.3 Palvelimet ja tiedonsiirto

Firebasen palvelimet toimivat CDN periaatteella ja tieto tallennetaan SSD-levyille. CDN tulee sanoista *Content Delivery Networks* ja tarkoittaa hajautettua palvelinkokonaisuutta, joka monistaa tallennettua tietoa useille eri palvelimille. Loppukäyttäjälle toimitetaan tieto sijainnin perusteella lähimmältä palvelimelta pienimmän mahdollisen vasteajan saavuttamiseksi (Marinescu 2013).

Tiedonsiirto tapahtuu ainoastaan 2048-bittisellä SSL-enkryptauksella https-protokollan yli. Suojaamattomalla yhteydellä tehtäviä kyselyitä ei Firebasella pysty tekemään ja ne aiheuttavat virheen.

2.3.4 Alustat ja kirjastot

Lähtökohtana palvelussa on tarjota sovelluskehittäjille mahdollisimman helposti lähestyttävä tekniikka. Palvellakseen mahdollisimman paljon kehittäjiä Firebase tarjoaa käytettäväksi valmiita kirjastoja useille eri alustoille. Kirjastojen avulla käyttäjä saa erittäin

pienellä työllä palvelun käyttöönsä. Näin kehittäjän kynnys ottaa palvelu käyttöön laskee huomattavasti.

Valmiita kirjastoja löytyy Androidille, iOS:lle, OSX:lle, Javalle sekä useille JavaScript-frameworkeille kuten AngularJS, EmberJS ja Backbone.js. Mikäli valmista kirjastoa ei löydy, voi Firebasen tietovarastoa hyödyntää REST-rajapinnan kautta. Tällöin palvelusta ei saa aivan kaikkea hyötyä irti, mutta itse tietoa pystyy joka tapauksessa sen kautta hyödyntämään.

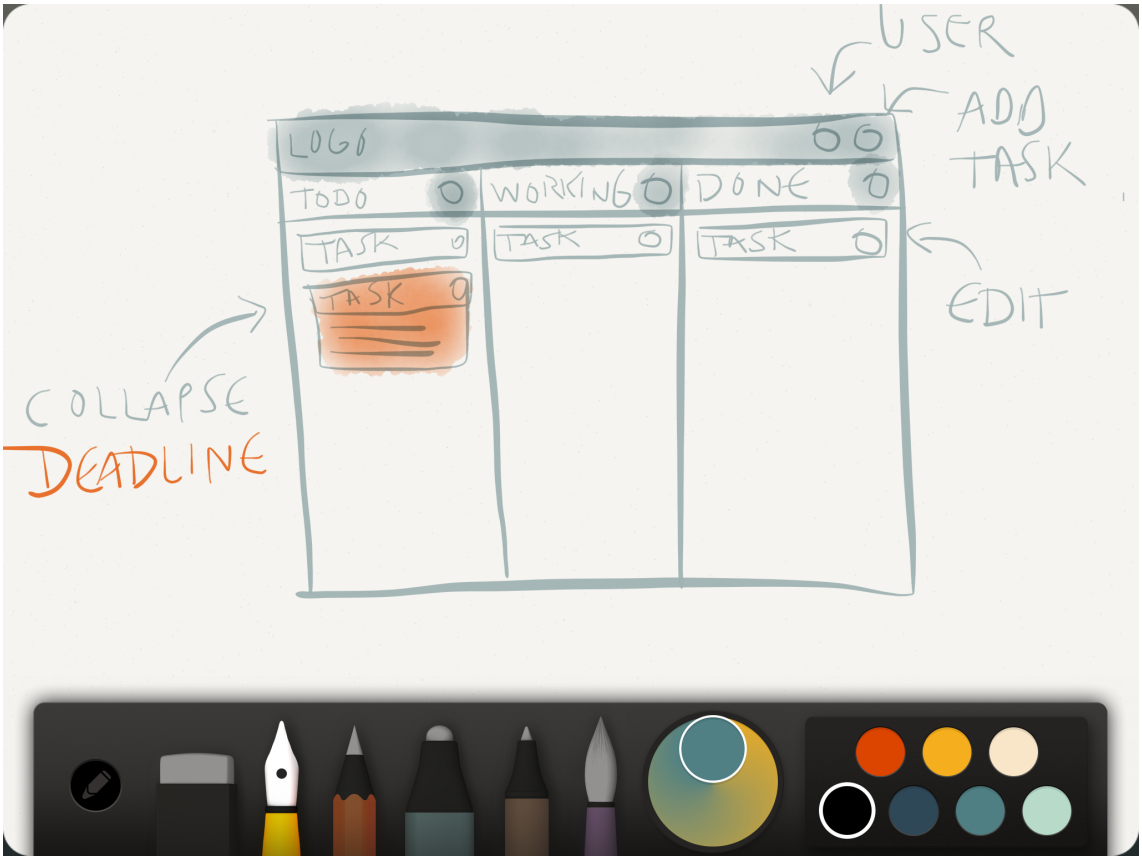
3 SOVELLUKSEN SUUNNITTELU

Työtehtävien hallinta-sovellusta suunnitellessani tein alustavan vaatimusmäärittelyn, jonka pohjalta toimeksiantajan edustajan kanssa teimme lopullisen vaatimusmäärittelyn. Sovelluksen tulisi olla responsiivinen ja sen tulisi toimia työpöytäkäytössä sekä Google Chromella että Mozilla Firefoxilla. Mobiilikäytössä sovelluksen tulisi toimia iPhonella ja iPadilla. Sovelluksessa tulisi olla käyttäjän tunnistautuminen. Käyttäjän tulisi pystyä luomaan, muokkaamaan ja poistamaan tehtäviä. Tehtäväkohtaisia tietoina tulisi olla otsikko, kuvaus, määräaika sekä työn vaihe.

Responsiivisuudella tarkoitetaan sovelluksen käyttöliittymän mukautumista käyttäjän päätelaitteen näyttökoon mukaan. Sovelluksen optimoidaan käytettäväksi puhelimilla, tableteilla ja tietokoneilla, jolloin sama sisältö saadaan kunkin laitteen näytölle sopivaksi käyttöliittymäksi.

3.1 Käyttöliittymä

Käyttöliittymä suunniteltiin yhdessä toimeksiantajan edustajan kanssa. Näin varmistettiin se, että sovellus vastaa käytettävyydeltään toimeksiantajan toiveita. Palaverissa piirrettiin rautalankamallit sovelluksen eri näkymistä (ks. kuva 3). Käyttöliittymän toivottiin myös olevan mahdollisimman selkeä ja minimalistinen.

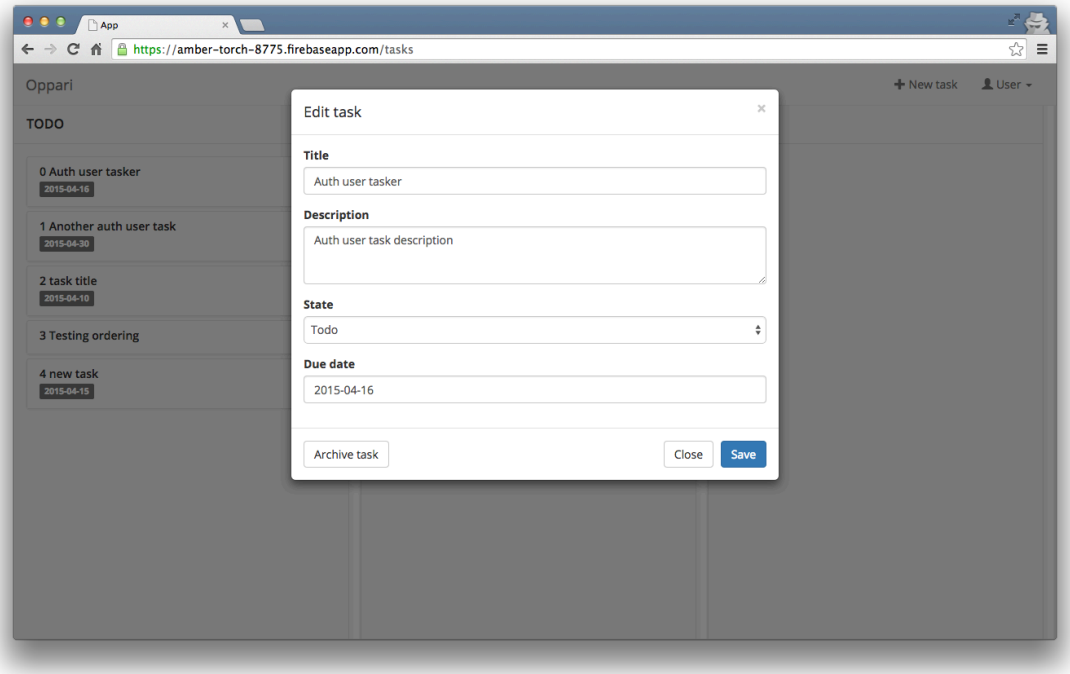


KUVA 3. Suunnittelupalaverissa piirretty rautalanka sovelluksen päänäkymästä

3.2 Rakenne

Sovelluksella on kaksi näkymää: kirjautuminen ja tehtävät. Tehtävien ja käyttäjän tietojen muokkaukset tapahtuu Bootstrapin Modal-elementissä. Modal on sivun päälle aukeava ikkuna (ks. kuva 4). Taakse jäävän sisällön päälle muodostuu tummentava verho. Modalin saa suljettua painamalla joko oikean yläkulman ruksia tai alaosasta löytyvää Cancel-painiketta. Myös tummentavan verhon painaminen sulkee Modalin.

Tehtävät-näkymä mukailee kanban-taulua, jossa on sarakkeita tehtävän eri vaiheille. Sarakkeiden määrä, eli työvaiheet, vaihtelevat prosessin tarpeiden mukaan. Toimeksiantajan tarpeisiin riitti kolme vaihetta: työjono, keskeneräinen ja valmis. Minkä tahansa vaiheen tehtäviä on mahdollista arkistoida.



KUVA 4. Kuvankaappaus tehtävän muokkauksen Modal-näkymästä.

3.3 Tekniikat

Sovelluksen frontend toteutetaan AngularJS-frameworkia hyödyntäen JavaScriptilla ja backend toteutetaan Firebasella. Näiden välillä apuna toimii Firebasen AngularJS–kirjasto AngularFire. Valitsin AngularJS:n koska olen käyttänyt sitä aikaisemmissa projekteissani ja se oli minulle jo entuudestaan tuttu.

Käyttöliittymä toteutetaan Bootstrapia hyödyntäen. Tämä vähentää sovelluksen kehitykseen vaadittua aikaa merkittävästi Bootstrapin tarjotessa valmiiksi suuren osan käyttöliittymän elementtien muotoilusta ja toiminnallisuudesta. Myös responsiivisuus saadaan toteutettua Bootstrapin avulla.

Käyttöliittymässä käytetään myös muutamia AngularJS-liitännäisiä sekä jQueryUI–kirjastoa. Käytetyillä kirjastoilla ja liitännäisillä saadaan tiettyjä toiminnallisuuksia valmiina.

Liitännäisten hallinnassa käytin pakettienhallintatyökalua nimeltä Bower, joka on tarkoitettu pääasiassa frontend-kehitykseen. Bower mahdollistaa sovelluksen liitännäisten

asentamisen, päivittämisen ja poistamisen komentoriville annettavilla komennoilla. Bowerin asennus ja käyttö vaatii Node.js:n ja npm:n lisäksi myös Git:n asennukset.

Node.js on avoimen lähdekoodin JavaScript-ajoympäristö palvelinpuolelle ja npm on sen mukana tuleva pakettienhallintatyökalu (npm 2015). Git on hajautettu versionhallintaohjelmisto (Preißel 2014).

4 SOVELLUKSEN TOTEUTUS

Sovelluksen toteutus tapahtui alkuun eri osioissa, jotka myöhemmässä vaiheessa yhdistettiin toimimaan yhdessä.

4.1 Sovelluksen runko

Sovelluksen runko rakentuu AngularJS-frameworkin ympärille. Tässä vaiheessa sovelluksella oli asennettuna frameworkin lisäksi Bootstrap. Näillä komponenteilla sain sovelluksen perustoiminnallisuuden toteutettua ja testattua.

AngularJS toimii lukemalla sivun HTML-elementteihin lisättyjen attribuuttien tietoja ja sitomalla ne kahdensuuntaisesti. Näin taustajärjestelmässä tapahtuvat tietojen muutokset muuttuvat suoraan käyttöliittymässä samoin, kuin käyttöliittymässä tapahtuvat muutokset on mahdollista tallentaa helposti taustajärjestelmän tietoihin. Perinteisestä staattisesta HTML-dokumentista saadaan toiminnallinen verkkosovellus.

4.2 Käyttöliittymä

Toimeksiantajan kanssa piirretyt rautalankamallit antoivat hyvän lähtökohdan käyttöliittymän tekoon. Rautalankamallista tein ensin karkeat html-versiot, jotka toimivat käyttöliittymän rakenteellisena pohjana. Perusrakenteen valmistuttua lisäsin Bootstrapin sivulle. Bootstrapista käytin versiota, jossa tyylitiedostot on valmiiksi tehty Sass:lla. Sass on sovelluksen tyylittelyyn käytettävä kieli, jolla pyritään vähentämään kirjoitettavien tyylimääreiden määrää. Syntaksiltaan Sass muistuttaa normaalia CSS-tyylitiedostoa, mutta se sallii muun muassa muuttujien käytön sekä sisäkkäiset elementit. Ennen varsinaista käyttöä Sass-tiedostot käännetään normaaleiksi CSS-tiedostoiksi (Sass 2015).

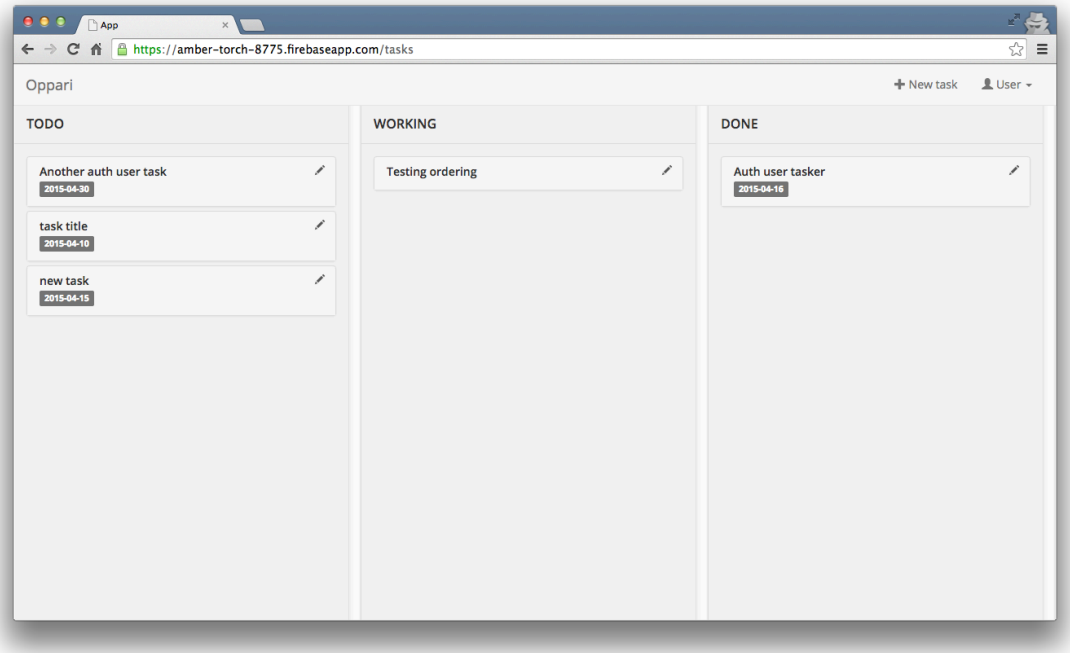
Sass:n kaltaisia työkaluja on markkinoilla muitakin kuten Less ja Stylus. Tämän sovelluksen toteutukseen valitsin Sass:n, koska olen käyttänyt sitä työelämässä ja todennut sen toimivaksi ja sopivaksi tähän käyttöön.

Bootstrapin toimiessa sovelluksessa aloitin osioiden rakentamisen lopulliseen muotoonsa käyttämällä Bootstrapin eri komponenttejä. Sovelluksen yläosassa käytin Navbar-komponenttia, jolla sain helposti toimivan ja käyttöliittymän kannalta selkeän navigaatiopalkin. Palkkiin lisäsin napin uuden tehtävän lisäämiselle sekä alasvetovalikon, johon lisäsin linkit käyttäjän tietojen muokkaamiseen sekä uloskirjautumiseen.

Uuden tehtävän lisäyksen sekä tehtävän ja käyttäjän tietojen muokkaukseen käytettävät tiedot aukeavat Bootstrapin Modal-komponenttiin. Tein sovellukselle yhden Modalin, jonka sisällä näytetään kussakin tapauksessa tarvittavat tiedot. Näin sain pidettyä sovelluksen html-rakennetta siistimpänä ja helpompana ylläpitää.

4.2.1 Päänäkymä

Päänäkymään tein kolme saraketta kuvastamaan kanban-taulun eri työvaiheita, yhden kullekin vaiheelle (ks. kuva 5). Syötetyt tehtävät laitoin työvaiheiden perusteella näkymään omissa sarakkeissaan. Jokaisesta tehtävästä tulee oma elementtinsä, jota pystyy raahaamalla siirtämään, sekä oman sarakkeensa sisällä että sarakkeesta toiseen, jolloin myös tehtävän vaihe päivittyy.

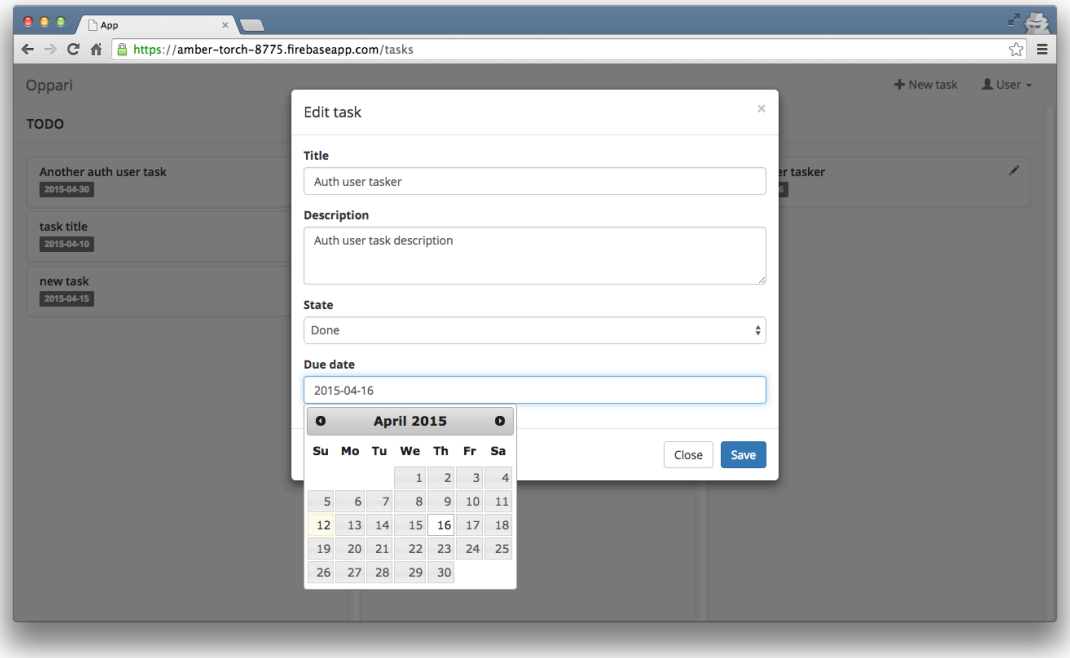


KUVA 5. Kuvankaappaus sovelluksen päänäkymästä.

Tehtävän elementissä näkyy tehtävän otsikko, mahdollinen määräaika sekä ikoni, jota painamalla tehtävän muokkausnäkymä aukeaa. Painamalla tehtävän otsikkoa avautuu näkymään tehtävän kuvausteksti. Tämän toteutin Bootstrapin Collapse-komponentilla, jossa piilossa oleva osuus kasvaa auki ja tuo elementin sisällön näkyville.

4.2.2 Tehtävien muokkaus

Muokkausnäkymässä on kentät tehtävän otsikolle ja kuvaukselle. Tehtävän tila valitaan alasvetovalikosta ja määräaikaa muokataan tekstikentän aktivoinnilla aukeavasta päivämäärän valitsimesta (ks. kuva 6). Tähän valitsimeen käytin GitHubista löytyvää käyttäjän *alexanderchan* tekemää *ui-date* –moduulia. Tämän moduulin valitsin löytämieni moduulivaihtoehtojen joukosta siksi, että olin käyttänyt sitä aikaisemmissa projekteissani ja kokemusteni mukaan se oli toiminnallisuuksiltaan, muokattavuudeltaan ja käyttöliittymältään sitä, mitä tarvitsin tähän sovellukseen. Pienenä miinuksena moduulille on sen riippuvuudet, sillä se vaatii AngularJS:n lisäksi jQuery- ja jQueryUI-kirjastot. AngularJS on mukana joka tapauksessa ja jQuery tarvitaan myös Bootstrapia käytettäessä, mutta jQueryUI joudutaan lataamaan ainoastaan tätä moduulia varten.



KUVA 6. Kuvankaappaus tehtävän muokkausnäkymästä, jossa määräajan muokkauskenttä aktiivinen.

Tehtävien vaihetta pystyy muuttamaan siirtämällä tehtävän sarakkeesta toiseen. Järjestyksen muokkaamiseen käytin GitHubista löytynyttä käyttäjän *kamilkp* tekemää AngularJS -moduulia *angular-sortable-view*. Tarkastelin myös vaihtoehtoja tälle moduulille, mutta tämä kyseinen oli ominaisuuksiltaan ehdottomasti paras tähän käyttötarkoitukseen. Moduuli oli pienikokoinen, eikä sillä ollut muita tarvittavia riippuvuuksia, kuin AngularJS–frameworkiin. Moduuli myös tuki sovelluksessa tarvittavaa kosketusnäytöllä raahaamista.

4.2.3 Käyttäjän tietojen muokkaus

Käyttäjän tietojen, kuten tehtävien, muokkaus tapahtuu avautuvassa Modalissa. Muokattavia käyttäjien tietoja ovat käyttäjänimi, etu- ja sukunimi ja sähköpostiosoite. Sovelluksen ollessa toimeksiantajan oma työkalu, ei tarvetta ylimääräisten tietojen keräämiselle ollut. Mikäli tilanne joistain syystä myöhemmin muuttuu, on käyttäjälle helppo lisätä muita tietoja. Tämä on yksi NoSQL-tietokannan eduista. Tietokannalla ei ole ennalta määritettyä mallia, joten mahdolliset myöhemmin lisättävät tietokentät eivät vaadi muutoksia tietokantaan, vaan frontend-kehittäjä pystyy suoraan määrittämään olemassa olevat lisätiedot.

4.3 Firebasen toiminnallisuus

Firebasen toiminnallisuuden aloitin perustamalla uuden sovelluksen Firebase-tiliin. Tilin luomalla sain yksilöllisen osoitteen sovelluksen tietovarastoon. Tämän jälkeen toin Firebase- ja AngularFire-kirjastot mukaan sovelluksen runkoon ja yhdistin ne luotuun Firebase-sovellukseen. Tiedon lataamista ja näyttämistä testasin lisäämällä testidataa Dashboardin kautta. Samalla sain myös testattua päivittyvän tiedon muutoksien toiminnallisuuden.

4.3.1 Tietojen haku ja näyttö

Tietojen hakua varten tein sovellukseen factoryn nimeltään Tasks, joka palauttaa taulukon kirjautuneen käyttäjän tehtävistä. Factory on AngularJS:n metodi, jolle määritetään

tietty tehtävä ja jota voi kutsua mistä controllerista tahansa. Tehtävänäkymän controller asettaa tuon Tasks-factorylta saadun taulukon $scope.tasks-arvoksi. $scope on AngularJS:n objekti, johon sovelluksen controllereista asetettu tieto on käytettävissä sovelluksen näkymissä.

Sovelluksen näkymässä pystytään näyttämään asetettua taulukkoa AngularJS ng-repeat-toiminnallisuudella, joka käy läpi kaikki tehtävät ja luo niistä jokaisesta oman elementin. Jokainen elementti on sidottu tietoriviin Firebasessa. Näin saadaan tietojen päivittyminen muuttamaan näkymää myös itse sovelluksessa.

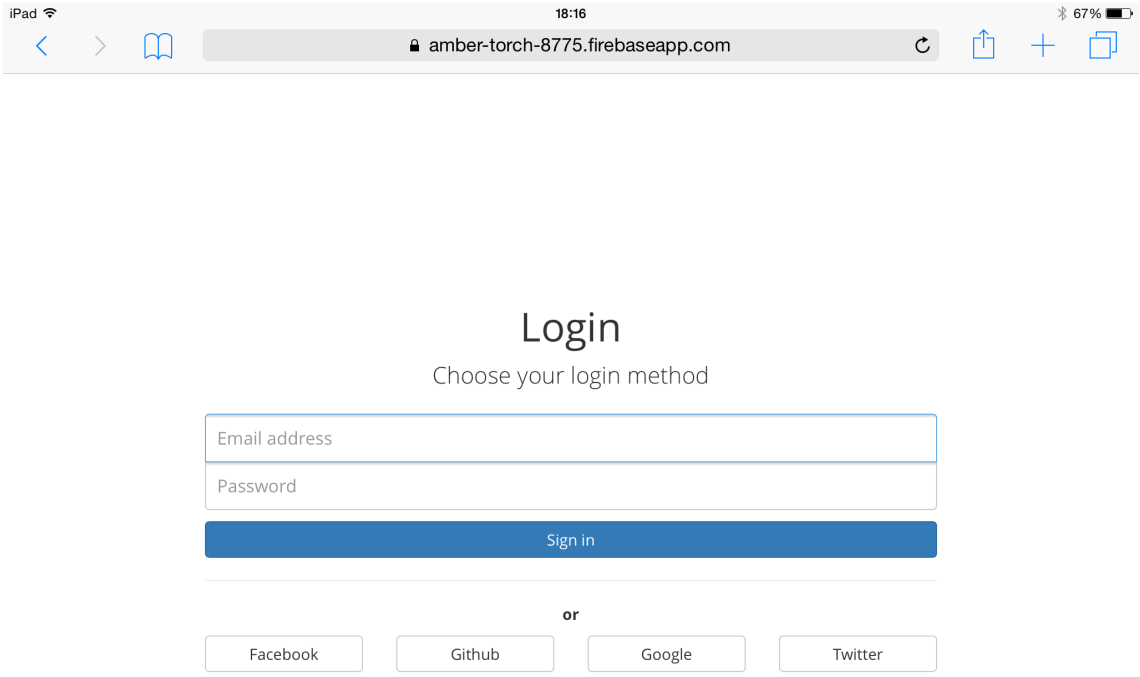
Firebasesta haettu tieto säilötään selaimen paikallisen tietovarastoon, josta se on muuttumattomana nopea käyttää uudelleen. Tietoja pystyy käyttämään myös ilman verkkoyhteyttä, jolloin tietojen päivitys tapahtuu vasta verkkoyhteyden toimiessa.

4.3.2 Tietojen tallennus

Sovelluksessa tapahtuva tietojen muokkaus muokkaa tiedot selaimen tietovarastoon sitä mukaa kun ne muuttuvat. Esimerkiksi tehtävän muokkauksessa tehtävät muutokset muuttavat päänäkyvässä olevaa tietoa. Tämä muutos ei vielä muuta tietoja Firebaseen, vaan ne päivittyvät ainoastaan selaimen tietovarastoon. Päivitys Firebasen tietovarastoon tapahtuu käyttäjän painaessa muokkauksen tallennusnappia. Tämä suorittaa AngularFiren $save-metodin, jolle annetaan parametrina itse tehtävän esiintymä.

4.4 Kirjautuminen

Käyttäjän tunnistamiseen käytin Firebasen tarjoamaa User Authenticationia, joka tarjoaa normaalin sähköposti-salasana-parin lisäksi kirjautumisen kolmansien osapuolten, kuten Facebook, Twitter, Github ja Google, tunnuksilla (ks. kuva 7).



KUVA 7. Kuvakaappaus sovelluksen kirjautumisnäkymästä iPadin Safarilla

AngularFiren kirjautumistoiminnoissa esiintyi sovelluksen toteuttamisen alkuvaiheissa dokumentaatiosta poikkeavaa toiminnallisuutta, eikä kirjautuminen toiminut. Firebase tarjosi kuitenkin päivitystä AngularFire–kirjastolleen ja tämän päivityksen myötä kirjautuminenkin toimi, kuten dokumentaatio sen kertoi toimivan.

Perinteisen sähköposti-salasana-parin toteutukseen Firebasesta löytyy valmiit toiminnallisuudet. Käyttäjän luontiin löytyy valmis metodi, jonka kautta käyttäjän syöttämät tiedot tallennetaan. Tämän toiminnon kautta luotujen käyttäjien tiedot tallennetaan kryptattuna normaalin tietovaraston ulkopuolelle. Kirjautumista varten löytyy myös valmis metodi, jonka parametreiksi annetaan käyttäjän syöttämät arvot. Firebase todentaa käyttäjän salattuja käyttäjätietoja vastaan.

Kolmansien osapuolen tunnuksilla kirjautumista varten kullekin palvelulle piti luoda kirjautumista varten oma sovellus, jonka tunnukset ja salaiset avaimet lisättiin Firebasen Dashboardiin. Tämä toiminnallisuus oli helppo ottaa käyttöön ja se toimi erittäin hyvin.

Käyttämällä Firebasen käyttäjienhallintaa vähensin omaa työmäärääni merkittävästi. Huolekseni jäi ainoastaan frontendin, eli käytännössä kirjautumissivun, luominen sekä kolmansien osapuolten sovellusten luominen ja tunnusten asettaminen.

4.5 Palvelintila

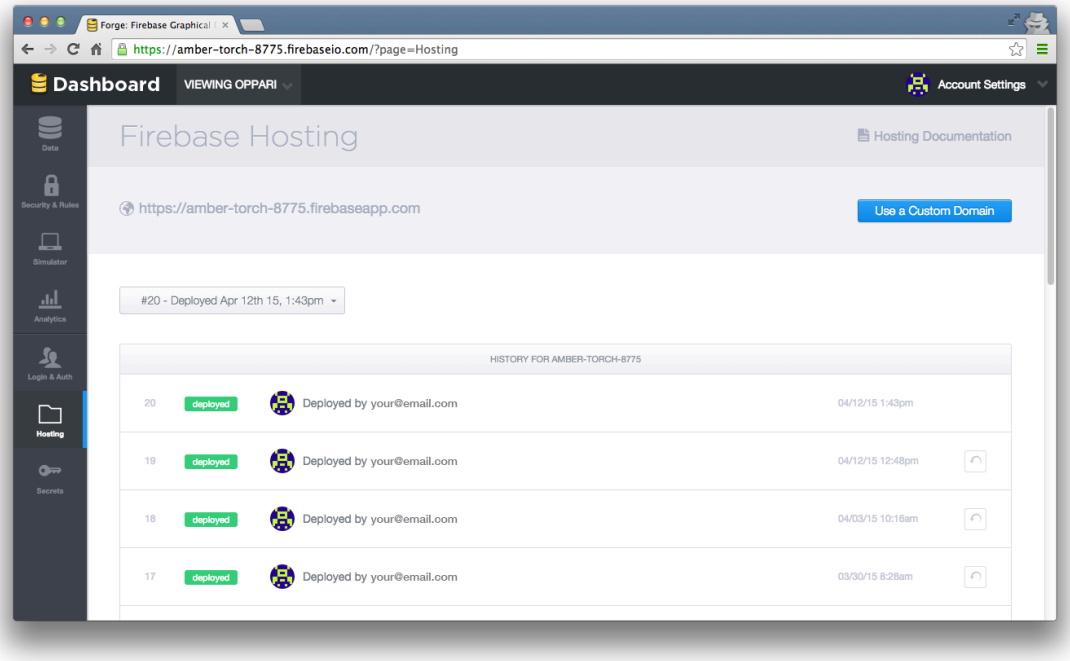
Sovelluksen kehittäminen tapahtui lokaalisti MAMP PRO -sovelluksen kautta toimivan Apache-palvelimen päällä. Tällä sain simuloitua tuotantoympäristöä vastaavan ympäristön, jonka kanssa kehittäminen ei vaatinut tiedostojen erillistä siirtämistä ulkoiselle palvelimelle jokaisen muutoksen jälkeen.

Tiettyjen vaiheiden jälkeen, käytännössä aina kun sain jonkun suuremman kokonaisuuden valmiiksi, siirsin sovelluksen varsinaiseen tuotantoympäristöön testattavaksi. Sovellus ei vaadi palvelimelta mitään erityistä ja se toimisi käytännössä millä tahansa palvelimella.

Tuotantoympäristöksi valitsin kuitenkin Firebasen tarjoaman Hosting-palvelun, jossa tiedot, kuten tietovarastokin, monistetaan usealle palvelimelle ja tarjoillaan käyttäjän sijainnin perusteella. Tämän palvelun valitsin kokeillakseni sen toimivuutta ja saadakseni kokemuksia, joita voi mahdollisesti hyödyntää myöhemmässä vaiheessa.

Hosting-palvelun käyttöönotto oli todella helppoa ja dokumentaatio tarjosi siihen erittäin hyvät ohjeet. Palvelua varten piti asentaa *firebase-tools* paketti. Paketin asennus vaatii, kuten muidenkin pakettien hallintaan käyttämäni Bower, Node.js ja npm asennukset koneelle. Tämän jälkeen palvelu alustettiin syöttämällä tietokoneen terminaalissa komento *firebase init*. Tämän jälkeen komennolla *firebase deploy* sovelluksen tiedostot siirtyivät palvelintilaan. Myöhemmässä vaiheessa tuolla samalla komennolla saa päivitettyä muuttuneen sovelluksen tiedostot kerralla ajan tasalle.

Firebase Dashboardissa olevan deploy-historian kautta näkee kaikkien Firebaseen ajettujen sovellusversioiden historian (ks. kuva 8). Dashboardin kautta pystyy määrittämään mitä versiota sovelluksesta ajetaan.



KUVA 8. Kuvankaappaus Firebasen deploy-historia näkymästä.

Hosting-palveluun saa syötettyä staattisia tiedostoja kuten html, css, javascript ja kuvatiedostot. Palvelu ei tarjoa tulkkeja eri kielille kuten PHP ja Python. Opinnäytetyösovellukseni kanssa tämä ei aiheuttanut ongelmia, koska sovelluksen tiedostot ovat kaikki staattisia. Sama tilanne pätee muidenkin AngularJS:n ja Firebasen kanssa rakennettujen sovellusten kanssa. Koska taustalogiikka tapahtuu palveluna muualla, ei tarvita kuin staattisia tiedostoja.

4.6 Osioiden yhdistäminen

Sovelluksen eri osioiden valmistuttua aloitin niiden yhdistämisen valmiiksi sovellukseksi. Ensiksi yhdistin sovelluksen rungon ja käyttöliittymän. Tämän jälkeen toin sovellukseen Firebasen toiminnallisuuden. Tässä vaiheessa laitoin näytettävän tiedon tulemaan Firebasen tiedoista, joka tässä vaiheessa oli ainoastaan Dashboardin kautta lisäämääni testidataa.

Käyttäjään sidottavan tiedon mahdollistamiseksi kirjautumisen toiminnallisuuden liittäminen sovellukseen oli seuraava vaihe. Tämän jälkeen uuden tiedon lisäys ja olemassa olevan tiedon muokkaus ja poisto toimi käyttäjäkohtaisesti ja sovelluksella oli mahdol-

lista käyttää sovelluksen kautta luotua tietoa. Käytettävä tieto koski niin käyttäjän tehtäviä, kuin käyttäjän omia tietoja.

Sovelluksen rakentaminen eri osissa oli minulle entuudestaan tuttua aikaisemmista työelämän projekteista, joissa sovelluksen eri osioita on usein tehnyt eri ihmiset. Osioiden yhdistämisessä ei esiintynyt ongelmia, vaan kaikki osat toimivat yhdessä toivotusti.

4.7 Testaus

Sovelluksen vaatimusmäärittelyn mukaan sovelluksen tulisi työpöytäkäytössä toimia Google Chrome- ja Mozilla Firefox -selaimilla ja mobiilikäytössä, sekä iPhonella että iPadilla.

Sovelluksen kehitys tapahtui Google Chromella, joten sen testaus tapahtui sovelluksen kehityksen lomassa. Mozilla Firefoxilla oli kuitenkin ongelma, jonka seurauksena päänäkymässä olevat sarakkeet eivät jakautuneet koko päänäkymän leveydelle, vaan ne kasautuivat näkymän vasempaan reunaan.

Ongelman aiheutti virhe käytetyn lisäosan vaatimassa html-elementtirakenteessa. Tämä aiheutti sen, että lisäosa ei laskenut kullekin sarakkeelle varattavaa leveyttä oikein. Muuttamalla sarakkeen html-rakennetta sain sovelluksen toimimaan oikein myös Mozilla Firefoxilla.

Mobiilikäyttöä testasin iPhone 5S:n sekä iPad:n Safari- ja Google Chrome -selaimilla. Näiden kohdalla ongelmaksi koitui Chromen popup-esto, jonka seurauksena kirjautuminen kolmannen osapuolien tunnuksilla ei ollut mahdollista.

Korjaukseksi tähän ongelmaan löysin AngularFirestä korvaavan metodin, joka kirjautumisruudun aukeamisen popup-ikkunaan sijasta uudelleenohjaa käyttäjän erilliselle kolmannen osapuolen kirjautumissivulle. Kirjautumisen jälkeen käyttäjä ohjataan takaisin sovellukseen. Tämän jälkeen kirjautuminen onnistui myös Chromella.

Rajallisen ajan vuoksi jouduin rajaamaan yksikkötestauksen pois tämän sovelluksen kehityksestä. Mikäli sovellusta jossain kohdassa ryhdyttäisiin jakamaan laajemmin kuin

ainoastaan toimeksiantajan käyttöön, tulisi testaus suorittaa vielä ennen sovelluksen laajempaa käyttöönottoa.

5 POHDINTA

Onnistuin opinnäytetyön tavoitteessani kiitettävästi. Sain hyvän kuvan verkkosovelluksen toteuttamisesta Firebasella. Ymmärrän nyt myös paremmin Firebasen tuomia mahdollisuuksista nykyaikaisessa sovelluskehityksessä. Opinnäytetyön toimeksiantaja oli myös erittäin tyytyväinen tekemääni sovellukseen, joka myös otettiin päivittäisen käyttöön. Sovellus on osoittanut heti hyödyllisyytensä muistilappujen kadotessa toimiston yleisilmeestä.

Ajankäyttöni opinnäytetyössä onnistui myös erittäin hyvin. Aikatauluni oli tiukka ja sovittaminen päivätyön ohelle haastavaa ja ajoittain henkisesti todella rankkaa. Onnistuin kuitenkin pysymään suunnitellussa aikataulussa eikä työmäärä ylittynyt. Sovelluksen toteutuksen osalta pystyin jopa alittamaan työmääräarvion.

Hyvällä suunnittelulla säästyin suuremmilta ongelmilta. Ainoa ongelma, jonka kanssa aikaa vierähti reilusti, oli kirjautumiseen liittyvä virhe. Tämä osoittautui kuitenkin käyttämäni kirjaston virheeksi. Kirjaston versiopäivityksellä ongelma kuitenkin ratkesi.

Hyödyin opinnäytetyöstäni erittäin paljon. Sen lisäksi että sain paljon uutta tietoa Firebasesta ja sen käytöstä, opin myös uutta AngularJS-frameworkista. Firebasen tarjoamat omaisuudet pienentävät kehittäjän työtä huomattavasti ja sovelluskehitys nopeutuu.

Sekä Firebasesta että AngularJS:stä saamiani kokemuksia pystyn ja tulen hyödyntämään työelämässä. Lisääntynyt osaamiseni tulee myös lisäämään mahdollisuuksiani työmarkkinoilla.

LÄHTEET

AngularJS. AngularJS API Reference. Luettu 22.2.2015
https://docs.angularjs.org/api

AngularJS. AngularJS Developer Guide. Luettu 22.2.2015
https://docs.angularjs.org/guide

Ashmore S & Runyan K. 2014. Introduction to Agile Methods. Addison-Wesley Professional.

Drucker, B. Firebase Tutorial: Building a Realtime App with Firebase. Luettu 22.2.20115
https://www.airpair.com/firebase/posts/firebase-building-realtime-app

Firebase. AngularFire API dokumentaatio. Luettu 22.2.2015
https://www.firebase.com/docs/web/libraries/angular/api.html

Firebase. AngularFire Development Guide. Luettu 22.2.2015
https://www.firebase.com/docs/web/libraries/angular/guide.html

Harrop, R., Ho, C. & Schaefer, C. 2014. Pro Spring, Fourth Edition. Apress.

Marinescu, D.C. 2013. Cloud Computing. Morgan Kaufmann.

Mehta, B. 2014. RESTful Java Patterns and Best Practices. Packt Publishing.

Npm 2015. Npm dokumentaatio. Luettu 22.2.2015
https://docs.npmjs.com/

Preißel R & Stachmann B. 2014. Distributed Version Control-Fundamentals and Workflows. Brainy Software.

Sass 2015. Sass dokumentaatio. Luettu 18.4.2015
http://sass-lang.com/documentation/

Vinci, R. 2014. AngularJS Web Application Development Blueprints. Packt Publishing.

LIITTEET

Liite 1. Ohjeet Firebasen ja AngularFiren käyttöönottoon ja peruskäyttöön

Firebase otetaan käyttöön muutamalla rivillä koodia.

```
<script src="https://cdn.firebase.com/js/client/2.1.2/firebase.js"></script>
<script src="https://cdn.firebase.com/libs/angularfire/0.9.2/angularfire.min.js"></script>
<script>
  var ref = new Firebase("https://<your-firebase>.firebaseio.com/");
</script>
```

Ladataan Firebase- sekä AngularFire-kirjastot liittämällä script-tagit html-tiedoston head-osaan. Alustetaan Firebase-objekti ref-muuttujaan.

Tietoa saa tallennettua $add-metodilla.

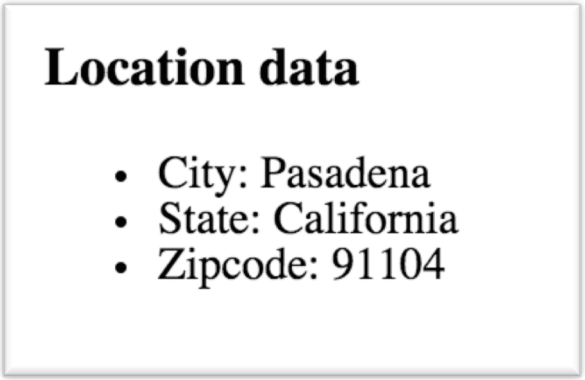
```
<script>
  $scope.data = $firebaseArray(ref);
  $scope.data.$add({
    title: "My Big Bang Data",
    author: "Sheldon Cooper",
    location: {
      city: "Pasadena",
      state: "California",
      zip: 91104
    }
  });
</script>
```

Metodilla $firebaseArray saadaan ladattua tietovaraston tiedot taulukkoon. $add-metodilla saadaan lisättyä parametrissä annettu objekti tietovarastoon.

$scope.data tiedot esitetään AngularJS:n näkymässä seuraavasti.

```
<h3>Location data</h3>
<ul>
  <li>City: {{ data.location.city }}</li>
  <li>State: {{ data.location.state }}</li>
  <li>Zipcode: {{ data.location.zip }}</li>
</ul>
```

$scope.data arvoja sidotaan näkymään aaltosulkujen sisällä ja niiden näyttämisestä huolehtii AngularJS.



Näkymä selaimessa annetuilla tiedoilla.